Codac2: Python manual

Luc Jaulin

1 Codac2

Codac2 https://codac.io/ is the version 2 of Codac. It can be used with C++ or Python. Codac2 is a library for interval analysis and constraint programming. It allows solving nonlinear problems while guaranteeing that all solutions are enclosed within computed bounds. This document, inspired by the documentation in https://codac.io/, focuses on practical usage through Python examples.

2 Basic interval manipulation

2.1 Intervals

Interval. An interval is defined by its lower and upper bound. For instance, to define the interval $x = [-2, 4]$, write:

```
x=Interval(-2,4)
```

Note that the lower bound must be smaller than upper bound. The statement:

```
y=Interval(5)
```

defines the degenerated interval (or singleton) $y = [5, 5] = \{5\}$. To define the interval $z = \mathbb{R} = [-\infty, \infty]$, write

```
z=Interval(-oo,oo)
```

The basic methods for intervals are `lb()` which returns the lower bound, `ub()` which returns the upper bound, `diam()` which returns the width, `mid()` which returns center and `is_empty()` which returns true if empty.

```
from codac import *
x = Interval(1,3)
y = Interval(-2,4)
a = x+y
```

```
b = x*y
print("x:", x)
print("y:", y)
print("x + y:", a)
print("x * y:", b)
```

This example returns

$$\begin{array}{rcccl} [a] & = & [x]+[y] & = & [-1,7] \\ [b] & = & [x]*[y] & = & [-6,12] \end{array}$$

2.2 Interval Vectors (Boxes)

An `IntervalVector` (also called a box) is a Cartesian product of intervals. It can be defined from a 2D array or a tuple. The use of boxes is illustrated by the following program

```
from codac import *
X = IntervalVector([[0,2],[1,3]]) # Create a 2D box
print("Box X:", X)
print("x1:", X[0]) # First component
print("x2:", X[1]) # second component
```

The execution of the program yields

$$\begin{array}{rcl} [\mathbf{x}] & = & \begin{pmatrix} [0,2] \\ [1,3] \end{pmatrix} \\ [x_1] & = & [0,2] \\ [x_2] & = & [1,3] \end{array}$$

2.3 Interval Matrix

An `IntervalMatrix` is matrix of intervals. For instance, to create the interval matrix

$$\mathbf{M} = \begin{pmatrix} 1 & [2,3] \\ [4,5] & [6,7] \end{pmatrix}$$

write

```
J=IntervalMatrix([[Interval(1,1),Interval(2,3)],[Interval(4,5),Interval(6,7)]])
```

2.4 Evaluating a Nonlinear Function

Elementary interval functions. Basic functions for real numbers such as `sin`, `cos`, `tan`, `acos`, `asin`, `atan`, `log`, `exp` are extended to intervals. For instance:

```
sin(Interval(0,3))
```

returns the interval $[0, 1]$.

The function

$$f(\mathbf{x}, y, \mathbf{m}) = \sqrt{(x_1 - m_1)^2 + (x_2 - m_2)^2} - y \tag{1}$$

can be defined as follows

```
y = ScalarVar()
x,m = VectorVar(2),VectorVar(2)
f = AnalyticFunction([x,y,m],sqrt(((x[0]-m[0])**2)+((x[1]-m[1])**2))-y)
```

For a vector function, the syntax slightly changes. For instance

$$f(\mathbf{x}, \mathbf{y}, \mathbf{m}) = \begin{pmatrix} x_1 + y_1 \cos(x_3 + y_2) - m_1 \\ x_2 + y_1 \sin(x_3 + y_2) - m_2 \end{pmatrix} \tag{2}$$

```
x,y,m = VectorVar(3),VectorVar(2),VectorVar(2)
f = AnalyticFunction([x,y,m],
   [x[0]+y[0]*cos(x[2]+y[1])-m[0],
   x[1]+y[0]*sin(x[2]+y[1])-m[1] ])
```

The evaluation is made with the method `eval` as shown by the following example:

```
from codac import *
x = ScalarVar()
f=AnalyticFunction([x],x**2+sin(x))
a=Interval(1,2)
y=f.eval(a)
print("y=", y)
```

The result $[y] = [1.84147, 5]$ encloses all values of $f(a)$ for $a \in [1, 2]$.

3 Contractors

3.1 Forward-backward contractor

The forward-backward contractor associated to

$$f(\mathbf{x}, y, \mathbf{m}) = 0$$

where f is defined as (1) is defined by

```
C = CtcInverse(f,0)
```

This contractor can be called with the method `.contract()`. It will contract all domains:

```
a = IntervalVector(2)
b = IntervalVector([[2,3],[5,6.2]])
d = Interval(4.5,5)
a,d,b = C.contract(a,d,b)
```

For the forward-backward contractor of a vector function, the principle is the same. For instance if, **f** is defined by (2), the contractor for

$$\begin{pmatrix} x_1 + y_1 \cos(x_3 + y_2) - m_1 \\ x_2 + y_1 \sin(x_3 + y_2) - m_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

is obtained as follows

```
C = CtcInverse(f,[1,2])
```

We now show by building a contractor for $x^2 = 4$.

```
from codac import *  # codac2
x = ScalarVar()
f=AnalyticFunction([x],x**2-4)
C=CtcInverse(f,0)
a=Interval(-10,10)
a=C.contract(a)
print("Contracted a:", a)
```

The result should be

$$[a] = [-2, 2]$$

3.2 Own contractor

To build your own contractor, you should build a class that inherits from the main class `Ctc_IntervalVector`. For instance if you want to build from scratch the contractor associated with the equation

$$x_1^2 + x_2^2 \in [4, 5]$$

and call the contractor in a paver, you can write the following program.

```
from codac import *
class myCtc(Ctc_IntervalVector):
   def __init__(C):
      Ctc_IntervalVector.__init__(C,2)
   def contract(C, X):
      x,y=X[0],X[1]
      r2=Interval(4,5)
      x2,y2=sqr(x),sqr(y)
      AddOp.bwd(r2,x2,y2)
      SqrOp.bwd(x2,x)
      SqrOp.bwd(y2,y)
      return X
C1= myCtc()
DefaultFigure.pave([[0,3],[0,3]], C1, 1e-2)
```

Since this contractor inherits from `Ctc_IntervalVector`, you will be able to compose `C1` with other contractors.

3.3 Own flattened contractor

A flattened contractor is a contractor whose input/output of which is a list of intervals (instead of a box). To build a your own flattened contractor, you should build a class for it. For instance, assume that you want to build a contractor for the distance equation

$$x^2 + y^2 = d^2$$

where x, y, d are the variables the domains of which should be contracted. You should first build the associated class

```
from codac import *
class my_flattenedcontractor():
   def __init__(C):
      x=VectorVar(3)
      f=AnalyticFunction([x],(x[0])**2+(x[1])**2-x[2]**2)
```

```
        C.Cm=CtcInverse(f,0)
    def contract(C,x,y,d):
        X=IntervalVector(3)
        X[0],X[1],X[2]=x,y,d
        C.Cm.contract(X)
        x,y,d =X[0],X[1],X[2]
        return x,y,d
```

Then, you create your contractor:

```
Cdist=my_flattenedcontractor()
```

and you use it as follows:

```
X,Y,D=Interval(0,5),Interval(0,5),Interval(6,10)
X,Y,D=Cdist.contract(X,Y,D)
```

Here, you get

```
X=[3.31662, 5], Y=[3.31662, 5] D=[6, 7.07107]
```

4 Separators

To initialize a separator from a function we use `SepInverse`. For instance a separator associated with the set

$$\mathbb{X} = \left\{\mathbf{x} | x_1^2 + x_2^2 \in [0, 80]\right\}$$

is build as follows.

```
from codac import *
x = VectorVar(2)
f = AnalyticFunction([x], sqr(x[0])+sqr(x[1]))
S = SepInverse (f, [1,2])
x0 = IntervalVector([[-3,3],[-3,3]])
p = pave(x0,S,0.01)
fig = Figure2D("A ring", GraphicOutput.VIBES)
fig.set_window_properties([50,50],[500,500])
fig.set_axes(x0)
fig.draw_paving(p,PavingStyle.blue_pink())
```

4.1 Operations on separators

4.1.1 Basic

The intersection, the union and the complement of separators is obtained using & , | and $\sim$.

For instance if $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ are three separators, then, the separator $\mathcal{S} = (\mathcal{S}_1 \cap \mathcal{S}_2) \cup (\mathcal{S}_2 \cap \mathcal{S}_3) \cup \left(\mathcal{S}_1 \cap \overline{\mathcal{S}_3}\right)$ is obtained by:

```
S=(S1&S2)|(S2&S3)|(S1&(~S3))
```

4.1.2 Relaxed intersection

If `L` is a list of separators, the q relaxed intersection of `L` is performed using `SepQInter`. For instance is $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ are three separators, we obtain the separator

$$\mathcal{S} = \bigcap^{\{1\}} \mathcal{S}_i$$

as follows:

```
q=1
L=[S1,S2,S3]
S=SepQInter(q,L)
```

4.1.3 Transformation

If $\mathbf{f}$ and $\mathbf{g}$ are two functions from $\mathbb{R}^n \to \mathbb{R}^n$ such that $\mathbf{g}$ is the reciprocal function of $\mathbf{f}$ and if $\mathcal{S}_1$ is a separator, then we define the transformation of $\mathcal{S}$ by $\mathbf{f}$ as follows

```
S2=SepTransform(S1,f,g)
```

If $\mathcal{S}_1$ is a separator for $\mathbb{S}_1$ then $\mathcal{S}_2$ is a separator for $\mathbb{S}_2 = \mathbf{f}(\mathbb{S}_1)$.

4.1.4 Own separator from two contractors

From two complementary contractors C_{in}, C_{out}, you can build a separator S. You should first build the associated class

```
class mySep(Sep):
   def __init(S):
      Sep.init__(S,2)
   def separate(S,Xin,Xout):
      Cout.contract(Xout)
      Cin.contract(Xin)
S=mySep()
```

4.2 Specific separators

4.2.1 SepWapper_IntervalVector

The optimal separator for the box $[\mathbf{x}]$ is defined as was defined as follows:

```
Sbox=SepWrapper_IntervalVector(X)
```

where X is the interval vector $[\mathbf{x}]$.

4.2.2 Polar

The optimal separator for the set

$$\mathbb{P} = \left\{(x, y) \in \mathbb{R}^2 \mid \exists(\rho, \theta) \in \mathbb{A} \text{ s.t. } x = \rho\cos\theta \text{ and } y = \rho\sin\theta\right\}$$

is defined as follows:

```
Sp=SepPolarCart(Sa)
```

where `Sa` is a separator for $\mathbb{A}$.

4.2.3 Projection

We can build a separator S2 associated to the projection of a set defined by the separator S1 using `SepProj`. The following example provides a separator S2 associated with the set

$$\left\{\mathbf{x} \in \mathbb{R}^2 \,|\exists \mathbf{a} \in [0,1]^2,\, (x_1 - a_1)^2 + (x_2 - a_2)^2 \in [4,9]\right\}$$

```
from codac import *
x = VectorVar(4)
f=AnalyticFunction([x], (x[0]-x[2])**2+(x[1]-x[3])**2)
S1=SepInverse(f,Interval(4,9))
S2 = SepProj(S1,[0,1],[[-1,1],[-1,1]],0.1)
DefaultFigure.pave([[-3,3],[-2,2]], S2, 5e-2)
```

In this example, S1 is a separator of dimension 4 whereas S2 is of dimension 2. In the same manner, we can build the projection for a contractor.

4.3 Sivia

`Sivia` admits as an input an initial box $[\mathbf{x}]$, a separator $\mathcal{S}$ (or a contractor), and an accuracy ε. For instance to run `Sivia` with an initial box `X`, a separator `S` for the set

$$\mathbb{X} = \{\mathbf{x} \in \mathbb{R}^2, (x_1 - 1)^2 + (x_2 - 2)^2 \in [4, 5]\}$$

and an accuracy `0.1`, write.

```
x=VectorVar(2)
f=AnalyticFunction([x],sqr(x[0]-1)+sqr(x[1]-2))
S=SepInverse(f,sqr(Interval(5,6)))
X0=IntervalVector([[-10,10],[-10,10]])
fig = Figure2D("Ring", GraphicOutput.VIBES)
fig.set_window_properties([50,50],[1000,1000])
fig.set_axes(X0)
fig.pave(X0,S,0.1,PavingStyle.blue_pink())
```

5 Tubes

5.1 Trajectories

We can build a trajectory from an analytical expression in t. For instance, to build to following trajectory

$$\mathbf{u}(t) = \begin{pmatrix} \sin(t) \\ \sin(2t) \end{pmatrix}$$

we first define the two components $u_1(t)$ and $u_2(t)$ and then concatenate them. This concatenation corresponds to the Cartesian product. We get the following program

```
from codac import *
dt = 0.01
tmax=3
t = ScalarVar()
f1=AnalyticFunction([t],sin(t))
f2=AnalyticFunction([t],sin(2*t))
u1=AnalyticTraj(f1,Interval(0,tmax)).sampled(dt)
u2=AnalyticTraj(f2,Interval(0,tmax)).sampled(dt)
u=traj_cart_prod(u1,u2)
```

5.2 Tubes

To create a tube, we can start with an AnalyticTraj and inflate it. The following program creates two tubes corresponding to

$$[\mathbf{u}](t) = \begin{pmatrix} [u_1](t) \\ [u_2](t) \end{pmatrix} = \begin{pmatrix} \sin(t) + [-0.1, 0.1] \\ \cos(2t) + [-0.1, 0.1] \end{pmatrix}$$

```
u1=AnalyticTraj(f1,Interval(0,tmax)).sampled(dt)
u2=AnalyticTraj(f2,Interval(0,tmax)).sampled(dt)
tdomain = create_tdomain(Interval(0,tmax),dt)
U1=SlicedTube(tdomain,u1).inflate(0.1)
U2=SlicedTube(tdomain,u2).inflate(0.1)
```

We can now compute primitives as follows:

```
θ=U1.primitive()
V=U2.primitive()
```

If we want to combine these new tubes, we have to use analytic functions. For instance, to create the 2 dimensional tube with components

$$\begin{array}{rcl} x_1(t) & = & \int_0^t v(\tau)\cos(\theta(\tau))d\tau \\ x_2(t) & = & \int_0^t v(\tau)\sin(\theta(\tau))d\tau \end{array}$$

we can write

```
θ1,V1 = ScalarVar(),ScalarVar()
φ1=AnalyticFunction([θ1,V1],cos(θ1)*V1)
φ2=AnalyticFunction([θ1,V1],sin(θ1)*V1)
dX1 = φ1.tube_eval(θ,V)
dX2 = φ2.tube_eval(θ,V)
X1= dX1.primitive()
X2= dX2.primitive()
X=tube_cart_prod(X1,X2)
DefaultFigure.draw_tube(X)
```

5.3 Tube contractor

5.3.1 From an analytical function

Consider trajectories $x_1(t)$,$x_2(t)$ that are linked by a constraint, say $x_1 - x_2 = 0$. These trajectories belong to tubes $[x_1](t), [x_2](t)$. Tube contractors contract the tubes without remaving any consistet values.

```
from codac import *
dt = 0.01
tmax=10
tdomain = create_tdomain(Interval(0,tmax),dt)
X1=SlicedTube(tdomain,Interval(2,5))
X2=SlicedTube(tdomain,Interval(1,3))
x1,x2 = ScalarVar(),ScalarVar()
ψ = AnalyticFunction([x1,x2],x2-x1)
ctc_ψ = CtcInverse(ψ,[0,0])
ctc_ψ.contract_tube(X1,X2)
```

In this case, we get two constant tubes given by $[x_1](t) = [x_2](t) = [2, 3]$.

5.3.2 Derivative contractor

Consider the constraint $\dot{x} = v$. The corresponding contractor is built as follows

```
V=SlicedTube(tdomain,...)
X=SlicedTube(tdomain,...)
C = CtcDeriv()
C.contract(X,V)
```

5.3.3 Dirac contractor

The tube is contracted for a single $t = t_1$. Assume that we know that at time t_1, $x(t_1) \in [1, 2]$, and that $x(t)$ is inside the tube X. We write:

```
X.set(Interval(1,2),t1)
```

This will contract X, without any computation. It just informs X that $x(t_1) \in [1, 2]$.

6 VIBES

The library VIBES is used only for drawing. It is now integrated in CODAC. To initialize a figure named fig1, write

```
fig1 = Figure2D("My figure", GraphicOutput.VIBES)
```

The window position and size is fixed on the screen as follows

```
fig1.set_window_properties([x1,y1],[width,height])
```

where `x1`, `y1` corresponds to the upper left corner (in pixel), `width`, `height` are also in pixel.

The frame is fixed by a box P as follows;

```
fig1.set_axes(P)
```

Draw a paving using SIVIA with a separator S with the initial box P, with a precision eps and the colors Blue-Pink (Blue for the outside and Pink for the inside):

```
fig1.pave(P,S,eps,PavingStyle.blue_pink())
```

You can draw a segment, a box, a disk, and a motorboat:

```
fig1.draw_line([[1,2],[3,4]],Color.black())
fig1.draw_box(IntervalVector([[2,3],[5,6]]),[Color.red(),Color.orange()])
fig1.draw_polygon([[0,1],[2,3],[4,5]],[Color.red(),Color.pink()])
fig1.draw_circle([1,2],3,[Color.black(),Color.yellow()])
fig1.draw_motor_boat(Vector([2,1,PI/6]),size=1,style=[Color.black(),Color.blue()])
```

To draw a tube $[\mathbf{x}](t)$, write

```
fig1.draw_tube(x,[Color.light_gray(),Color.light_gray()])
```

To draw a trajectory $\mathbf{x}(t)$, write

```
fig1.draw_trajectory(x)
```

7 Combining Codac with Sympy

Sometimes, you have a Sympy expression `E(x)` and you want to use SIVIA with this expression. A SymPy expression `E(x)` is internally represented as an Abstract Syntax Tree (AST) composed of immutable basic objects. In this tree, internal nodes are operations or functions (like Add, Mul, Pow, sin). Leaf nodes are atomic elements with no children (like Symbol, Integer, Rational). To use codac, you need to convert `E(x)` to an `AnalyticalFunction`. In the following program, the Sympy expression `E(x)` is first converted (via `sp2str(E)`) into a string by scanning the corresponding AST. Then, the string is interpreted by Python by the function to `sp2codac(x,E)`.

```
from codac import *  # Version 2
import sympy as sp
def sp2str(E):
    match E:
```

```
            case _ if E.is_Atom:
                return str(E)
            case sp.Add() | sp.Mul():
                op = " + " if isinstance(E, sp.Add) else " * "
                return "(" + op.join(sp2str(a) for a in E.args) + ")"
            case sp.Pow(exp=2):
                return f"sqr({sp2str(E.base)})"
            case sp.Pow():
                return f"({sp2str(E.base)})^({sp2str(E.exp)})"
            case _ if E.func in (sp.sin, sp.cos, sp.exp):
                return f"{E.func.__name__}({sp2str(E.args[0])})"
            case _:
                return ""
    def sp2codac(x,E):  # a sympy expression E(x) to AnalyticFunction of codac2
        return AnalyticFunction([x],eval(sp2str(E)))
    x = [sp.symbols(f'x[{i}]') for i in range(3)] #sympy world
    E1=(x[0]-1)**2+(x[1]-2)**2
    x=VectorVar(2) #sympy world
    f=sp2codac(x,E1)
    S=SepInverse(f,sqr(Interval(1,2)))
    X0=IntervalVector([[-10,10],[-10,10]])
    fig = Figure2D("Ring", GraphicOutput.VIBES)
    fig.set_window_properties([50,50],[1000,1000])
    fig.set_axes(X0)
    fig.pave(X0,S,0.01,PavingStyle.blue_pink())
```

8 Exercises of IAMOOC

We give here the solutions of the exercises 9, 10, 11 and 12 of IAMOOC

webperso.ensta.fr/jaulin/iamooc.htm

A zip file with these programs with CODAC2 are also here:

webperso.ensta.fr/jaulin/iamooc_ex_codac2.zip

8.1 Exercises 9: Contractors and Separators

Consider the two rings defined by

$$\begin{aligned}\mathbb{S}_1 &= \left\{(x,y)\in\mathbb{R}^2 \mid \ (x-1)^2+(y-2)^2 \in [4,5]^2\right\}\\ \mathbb{S}_2 &= \left\{(x,y)\in\mathbb{R}^2 \mid \ (x-2)^2+(y-5)^2 \in [5,6]^2\right\}.\end{aligned}$$

Build a separator for $\mathbb{S}_1 \cup \mathbb{S}_2$ and draw the corresponding paving.

Program

```
from codac import *
x=VectorVar(2)
f1=AnalyticFunction([x],sqr(x[0]-1)+sqr(x[1]-2))
f2=AnalyticFunction([x],sqr(x[0]-2)+sqr(x[1]-5))
S1=SepInverse(f1,sqr(Interval(4,5)))
S2=SepInverse(f2,sqr(Interval(5,6)))
S=S1|S2
X0=IntervalVector([[-10,10],[-10,10]])
fig = Figure2D("Codac2: Ring", GraphicOutput.VIBES)
fig.set_window_properties([50,50],[500,500])
fig.set_axes(X0)
fig.pave(X0,S,0.1,PavingStyle.blue_pink())
fig.draw_box(X0,[Color.black()])
```

8.2 Exercises 10: Parameter estimation

Consider the bounded-error parameter estimation problem defined by

$$p_1 \cdot e^{p_2 \cdot t} \in [y](t)$$

where $\mathbf{p} = (p_1, p_2)$ is the parameter vector, t is the time, $[y](t)$ is the output interval that is returned by a sensor. Assume that for different t, we collected the following interval measurements:

t_i	0.2	1	2	4
$[y](i)$	$[1.5, 2]$	$[0.7, 0.8]$	$[0.1, 0.3]$	$[-0.1, 0.03]$

Define the set of all parameter vectors consistent with the ith data interval is defined by

$$\mathbb{P}_i = \left\{ \mathbf{p} \in [-3,3]^2 \mid p_1 \cdot e^{p_2 \cdot t_i} \in [y](t_i) \right\}$$

where $[-3,3]^2$ corresponds a prior box which is known to enclose $\mathbf{p}$. For $q \in \{0,1,2\}$, compute the q-relaxed intersection

$$\mathbb{P}^{\{q\}} = \bigcap_i^{\{q\}} \mathbb{P}_i.$$

Program

```
from codac import *
T=[0.2,1,2,4]
```

```
Y=[Interval(1.5,2),Interval(0.7,0.8),Interval(0.1,0.3),Interval(-0.1,0.03)]
seps=[]
for i in range(0,4):
   p=VectorVar(2)
   fi=AnalyticFunction([p],p[0]*exp(p[1]*T[i]))
   seps.append(SepInverse(fi,Y[i]))
q=1
S=SepQInter(q,seps)
P=IntervalVector([[-10,10],[-10,10]])
fig = Figure2D("Codac2: Estimator", GraphicOutput.VIBES)
fig.set_window_properties([50,50],[500,500])
fig.set_axes(P)
fig.pave(P,S,0.1,PavingStyle.blue_pink())
fig.draw_box(P,[Color.black()])
```

8.3 Exercises 11: Localization of a robot

A robot has to localize inside an environment made of 4 landmarks $\mathbf{m}(i), i \in \{1,\ldots,4\}$. It is able to measure its distance $d(i)$ and the azimuth $\alpha(i)$ corresponding to each landmark with some bounded errors. The coordinates of landmarks and the interval measurements are given in the following table

i	$\mathbf{m}(i)$	$[d](i)$	$[\alpha](i)$
1	$(6,12)$	$[10,13]$	$[0.5,1]$
2	$(-2,-5)$	$[8,10]$	$\left[-3,-\frac{3}{2}\right]$
3	$(-3,10)$	$[5,7]$	$[1,2]$
4	$(3,4)$	$[6,8]$	$[2,3]$

The set of all positions $\mathbf{p} \in \mathbb{R}^2$ for the robot that are consistent with the ith measurements is

$$\mathbb{P}_i = \left\{(p_1,p_2) \in \mathbb{R}^2 \mid \exists d \in [d](i), \exists \alpha \in [\alpha](i) \text{ s.t. } m_1(i) = p_1 + d\cos\alpha \text{ and } m_2(i) = p_2 + d\sin\alpha\right\}.$$

Using the optimal separator named SepPolarCart, draw a subpaving approximation for the sets $\mathbb{P}_i$. Give the set of all $\mathbf{p}$ that are consistent with the largest number of data.

Program

```
from codac import *
Marks = [[6, 12], [-2, -5], [-3, 10], [3, 4]]
D= [Interval(10, 13), Interval(8, 10), Interval(5, 7), Interval(6, 8)]
Alpha = [Interval(0.5, 1), Interval(-3, -1.5), Interval(1, 2), Interval(2, 3)]
def SepPolarXY(ρ,θ): return SepPolarCart(SepWrapper_IntervalVector([ρ,θ]))
seps=[]
```

```
for i in range(0,4):
   m=Marks[i]
   v=VectorVar(2)
   p=VectorVar(2)
   fforw=AnalyticFunction([v],[m[0]-v[0],m[1]-v[1]])
   fback=AnalyticFunction([p],[m[0]-p[0],m[1]-p[1]])
   S1=SepPolarXY(D[i],Alpha[i])
   S2=SepTransform(S1,fforw,fback)
   seps.append(S2)
q=1
sep=SepQInter(q,seps)
P=IntervalVector([[-20,20],[-20,20]])
fig = Figure2D("Codac2: locpie", GraphicOutput.VIBES)
fig.set_window_properties([50,50],[1000,1000])
fig.set_axes(P) fig.draw_box(P,[Color.black()])
fig.pave(P, sep, 0.1,PavingStyle.blue_pink())
for m in Marks:
   fig.draw_circle(m,0.3,[Color.black(),Color.yellow()])
```

8.4 Exercises 12: Simultaneous Localization and Mapping

Consider a robot at position (x, y) moving inside an unknown environment. We assume that its motion is described by the discrete time state equations:

$$\begin{cases} x(k+1) &= x(k) + 10 \cdot \delta \cdot \cos\theta(k) \\ y(k+1) &= y(k) + 10 \cdot \delta \cdot \sin\theta(k) \\ \theta(k+1) &= \theta(k) + \delta \cdot (u(k) + n_u(k)) \end{cases}$$

where $k \in \{0, \ldots, 100\}$ is the discrete time, $\delta = 0.1$ corresponds to the sampling time, θ to the heading, u to the desired rotational speed and n_u to a noise. We assume that at time $k = 0$, we have $x(k) = y(k) = 0$ and $\theta(k) = 1$. The desired input $u(k)$ is chosen as

$$u(k) = 3 \cdot \sin^2(k\delta).$$

We assume that the heading is measured (using a compass for instance) with a small error:

$$\theta^m(k) = \theta(k) + n_\theta(k).$$

For all k we assume that $n_u(k)$ and $n_\theta(k)$ belong to $[-0.03, 0.03]$.

1) Simulate the system using a uniform random noise and draw an interval tube enclosing the trajectory.

2) In the environment, we assume that we have 4 landmarks, the coordinate of which are given by the following table.

i	0	1	2	3
$\mathbf{m}(i)$	$(6, 12)$	$(-2, -5)$	$(-3, 20)$	$(3, 4)$

We do not know these coordinates. For each k the robot is able to measure the distance to one of these landmarks (taken randomly), with an accuracy of ± 0.03. By simulation, generate a set of intervals containing these all collected distances with their uncertainties.

3) Using a contractor-based method, improve the accuracy for the enclosure for the robot while simultaneously enclosing all landmarks by boxes.

Program

```
from codac import *
import numpy as np
import random import math
def rand(I):
   x=I.lb()+random.random()*I.diam()
   return x
class ctc_state():
   def __init__(C):
      x,y,x1,y1,th=ScalarVar(),ScalarVar(),ScalarVar(),ScalarVar(),ScalarVar()
      fx=AnalyticFunction([x1,x,th],x1-x-dt*10*cos(th))
      fy=AnalyticFunction([y1,y,th],y1-y-dt*10*sin(th))
      C.Cx=CtcInverse(fx,Interval(0,0))
      C.Cy=CtcInverse(fy,Interval(0,0))
   def contract(C,x1,y1,x,y,th):
      X=IntervalVector(3)
      X[0],X[1],X[2]=x1,x,th
      C.Cx.contract(X)
      x1,x,th=X[0],X[1],X[2]
      X=IntervalVector(3)
      X[0],X[1],X[2]=y1,y,th
      C.Cy.contract(X)
      y1,y,th=X[0],X[1],X[2]
      return x1,y1,x,y
class ctc_marks():
   def __init__(C):
      x,y,mx,my,d=[ScalarVar() for _ in range(5)]
      f=AnalyticFunction([x,y,mx,my,d],(x-mx)**2+(y-my)**2-d**2)
      C.Cm=CtcInverse(f,Interval(0,0))
   def contract(C,x,y,mx,my,d):
      X=IntervalVector(5)
      X[:5]=x,y,mx,my,d
      C.Cm.contract(X)
```

```
      x,y,mx,my=X[:4]
      return x,y,mx,my

fig = Figure2D("Codac2: SLAM", GraphicOutput.VIBES)
fig.set_window_properties([50,50],[1000,1000])
fig.set_axes([[-20,20],[-20,20]])
_x,_y,_th,dt=0,0,1,0.1
_X,_Y=[],[]
Th,J,D=[],[],[]
_mx,_my=[6,-2,-3,3],[1,-5,4,4]
noise=0.03*Interval(-1,1)
kmax=100
for k in range(0,kmax):
   j=random.randint(0,3)
   J.append(j)     _
   d=np.sqrt((_mx[j]-_x)**2+(_my[j]-_y)**2)
   D.append(_d+rand(noise)+noise)
   _X.append(_x)
   _Y.append(_y)
   Th.append(_th+rand(noise)+noise)
   _x=_x+dt*10*np.cos(_th)
   _y=_y+dt*10*np.sin(_th)
   _th=_th+dt*(3*np.sin(k*dt)**2+rand(noise))
X=[Interval(-20,20)]*(kmax+1)
X[0]=Interval(0)
Y=X.copy()
Mx=[Interval(-20,20)]*4
My=[Interval(-20,20)]*4
cstate=ctc_state()
cmarks=ctc_marks()
for n in range(0,5):
   for k in range(1,kmax):
      X[k],Y[k],X[k-1],Y[k-1]=cstate.contract(X[k],Y[k],X[k-1],Y[k-1],Th[k-1])
   for k in range(kmax,1,-1):
      X[k],Y[k],X[k-1],Y[k-1]=cstate.contract(X[k],Y[k],X[k-1],Y[k-1],Th[k-1])
   for k in range(0,kmax):
      j=J[k]
      X[k],Y[k],Mx[j],My[j]=cmarks.contract(X[k],Y[k],Mx[j],My[j],D[k])
   for Xk,Yk in zip(X,Y):
      fig.draw_box([Xk,Yk],[Color.blue(),Color.blue()])
   for j in range(0,4):
      fig.draw_box([Mx[j],My[j]],[Color.pink()])
for Xk,Yk in zip(X,Y):
```

```
   fig.draw_box([Xk,Yk],[Color.black(),Color.blue()])
for j in range(0,4):
   fig.draw_box([Mx[j],My[j]],[Color.red(),Color.red()])
   fig.draw_circle([_mx[j],_my[j]],0.1,[Color.black(),Color.pink()])
for k in range(0,kmax):
   fig.draw_circle([_X[k],_Y[k]],0.1,[Color.black(),Color.green()])
```